Alexander Hay
ME333 - Mechatronics
Chapter 5, 3-4

3. a. float add, sub, mul ,div
doub add, sub, mul, div
long long add, sub, mul, div

C code:

```
f3 = f1 + f2;
printf("float addition: %f \r\n", f3);
```

Assembly code:

```
f3 = f1 + f2;
9d0041e8:   0f4011c2    jal   9d004708 <__addsf3>
9d0041ec:   00a08021    move  s0,a1
printf("float addition: %f \r\n", f3);
9d0041f0:   3c049d00    lui   a0,0x9d00
9d0041f4:   24844cd0    addiu a0,a0,19664
9d0041f8:   0f401802    jal   9d006008 <_printf_cdfFnopuxX>
9d0041fc:   00402821    move  a1,v0

9d004708 <__addsf3>:
9d004708:   00044dc2 312900ff 00055dc2
316b00ff.M....)1.]....k1
9d004718:   3c078000 00044200 01074025 00055200
      ...<.B..%@...R..
9d004728:   01475025 252dffff 2da100fe 10200061
      %PG...-%...-a. .
9d004738:   00000000
      ....
```

b. int add, sub, mul

C code:

```
f3 = f1 + f2;
printf("float addition: %f \r\n", f3);
```

Assembly code:

```
i3 = i1 + i2;
printf("int addition: %i \r\n", i3); // operation makes c3 an int
9d0040f4:   3c049d00    lui   a0,0x9d00
9d0040f8:   24844c0c    addiu a0,a0,19468
9d0040fc:   0f401802    jal   9d006008 <_printf_cdfFnopuxX>
9d004100:   02252821    addu  a1,s1,a1
9d0028e4:   304200ff    andi  v0,v0,0xff
```

The **`andi`** instruction does a bitwise AND of two 32-bit patterns. At run time the 16-bit immediate operand is padded on the left with zero bits to make it a 32-bit operand. [1]

3. c.

	char	int	long long	float	long double

+	3:2 (2)	1:1 (2)	1:1 (2)	J	J
-	1:1 (1)	2:1 (1)	2:1 (1)	J	J
*	1:1 (1)	2:1 (1)	2:1 (1)	J	J
/	1:1 (3)	1:1 (3)	1:1 (3)	J	J

* I used functions and I think it goofed up the ratios.

d.

Section	Address	Length (bytes)	(dec)
.text.dp32mul	0x9d0034b8	0x4b8	1208
.text.dp32subadd	0x9d00422c	0x430	1072
.text.dp32mul	0x9d00465c	0x32c	812
.text.fpsubadd	0x9d004c3c	0x278	632
.text.fp32div	0x9d004eb4	0x230	560
.text.fp32mul	0x9d005300	0x1bc	444

4. & uses 4 commands
| uses 4 commands
<< uses 3 commands
>> uses 3 commands

```
u3 = u1 & u2;
9d0027c0:   8fc30010    lw    v1,16(s8)
9d0027c4:   8fc20014    lw    v0,20(s8)
9d0027c8:   00621024    and   v0,v1,v0
9d0027cc:   afc20018    sw    v0,24(s8)

u3 = u1 | u2;
9d0027d0:   8fc30010    lw    v1,16(s8)
9d0027d4:   8fc20014    lw    v0,20(s8)
9d0027d8:   00621025    or    v0,v1,v0
9d0027dc:   afc20018    sw    v0,24(s8)

u3 = u2 << 4;
9d0027e0:   8fc20014    lw    v0,20(s8)
9d0027e4:   00021100    sll   v0,v0,0x4
9d0027e8:   afc20018    sw    v0,24(s8)

u3 = u1 >> 3;
9d0027ec:   8fc20010    lw    v0,16(s8)
9d0027f0:   000210c2    srl   v0,v0,0x3
9d0027f4:   afc20018    sw    v0,24(s8)
```

References:

[1] Kjell, Bradley. “Handy ANDI Instruction.” *Programmed Instruction to MIPS Assembly Language*, Central Connecticut State University, chortle.ccsu.edu/assemblytutorial/Chapter-11/ass11_11.html.

Alexander Hay
ME333 - Mechatronics
Chapter 6, 1, 4, 5, 8, 9, 16

1.

Pros	Cons
Polling isn't limited, but you can only have so many interrupts	Polling is slower
	Functions must be called in correct order

4.
a. CPU jumps to ISR
b. CPU will jump to level 4 ISR, complete it, then back to level 2 ISR
c. CPU will complete current subpriority 4.2 ISR before moving to on to 4.0
d. CPU will complete level 6 ISR before jumping to level 4

5.
a. Write ISR (clear IRQ flag)
Disable interrupts at CPU
Configure a device to generate IRQs
Set priority of ISR (IPCy)
Clear IRQ flag (IFSx)
Enable IRQ (IECx)
Enable interrupts at CPU

b. Changes the current set of registers to another shadow set. This avoids having to save and reuse them, saving clock cycles (running faster)

8.
a.
```
IEC0SET = 0x100;
IFS0CLR = 0x100;
IPC2CLR = 0x1F;
IPC2SET = 0x16;
```

b.
```
IEC1SET = 0x8000;
IFS1CLR = 0x8000;
IPC8CLR = 0x1F000000;
IPC8SET = 0x19000000;
```

c.
```
IEC2SET = 0x10;
ISF2CLR = 0x10;
IPC12CLR = 0x1F00;
IPC12SET = 0x1F00;
```

d.
```
IEC0SET = 0x800;
ISF0CLR = 0x800;
IPC2CLR = 0x1F00000;
IPC2SET = 0xE000000;
INTCON = 0x4;
```

9.
```
INTCONSET = 0x3;        // step 3: INT0 and INT1 trigger on rising edge
IPC0CLR = 0x1F000000;   // step 4: clear 5 priority and subp bits for INT0
IPC0SET = 0x18000000;   // step 4: set INT0 to priority 6 subpriority 0
IPC1CLR = 0x1F000000;   // step 4: clear 5 priority and subp bits for INT1
IPC1SET = 0x18000000;   // step 4: set INT1 to priority 6 subpriority 0
IFS0bits.INT0IF = 0;    // step 5: clear INT0 flag status
IFS0bits.INT1IF = 0;    // step 5: clear INT1 flag status
IEC0SET = 0x88;         // step 6: enable INT0 and INT1 interrupts
```

16. N/A – coming into office hours for help, assignment submitted before then. Code attached but button unresponsive

```
#include "NU32.h"           // constants, funcs for startup and UART
#define DELAYTIME 400000    // 400 thousand core clock ticks, or 0.01 second


void debounce(){
  _CP0_SET_COUNT(0);
  while(_CP0_GET_COUNT() < DELAYTIME) {
    ;
  }
}


void __ISR(_EXTERNAL_0_VECTOR, IPL2SOFT) Ext0ISR(void) { // step 1: the ISR
  _CP0_SET_COUNT(0);
  debounce();                // (wait 10ms)
  if (PORTDbits.RD0 == 0){   // read pin again
    NU32_LED1 = 1;           // LED1 and LED2 off; initialized as on
    NU32_LED2 = 1;
  }
  IFS0bits.INT0IF = 0;       // clear interrupt flag IFS0<3>
}
```